VIRAGE

VIDEO BROWSING USING COOPERATIVE VISUAL AND LINGUISTIC INDICES

Proposal No.: ARPA SB 961-037
Contract No.: DAAH01-96-C-R121

Phase 1: Final Project Report

Dr. Arun Hampapur
Dr. Amarnath Gupta
Dr. Ramesh Jain

Virage Incorporated
177 Bovet Road, Suite 520
San Mateo, CA 94402-3121

Video Browsing Using Cooperative Visual and Linguistic Indices

Proposal # ARPA SB 961-037
Contract # DAAH01-96-C-R121

Phase I: Final Project Report

Dr. Arun Hampapur (Principal Investigator) Dr. Amarnath Gupta
Dr. Ramesh Jain (Co Principal Investigator)

Virage, Inc.
177 Bovet Road
San Mateo, CA 94402

1 Introduction

Management of video information for content-based retrieval has been recognized as a significant problem in the multimedia community for a long time. Recent literature reports research activities in three major areas: segmentation of special video events based on both abrupt and gradual scene changes; motion analysis techniques applied toward determination of camera movement and object/background segmentation; and data-modeling and query-language development with the assumption that objects in the video have been segmented by low-level processing. While these approaches are all valid research problems, we as an industrial research group, have taken a more application-driven approach, and address the questions: "What is the purpose of retrieval by video content? How can we use both imagery-based and non-imagery-based information together to access large amounts of video by content?" The proposal was chartered to explore methods and develop techniques for information retrieval and browsing of news video using both imagery and linguistic content. We chose television news as the domain of this video retrieval project. The long-term goal for video news information retrieval/browsing system is to have the ability to:

1. isolate a specific information item using imagery, linguistic, and audio description and any combination of these
2. present it at different levels of abstraction (detail) using any a priori knowledge about the domain of application to the maximum extent possible
3. given one piece of information related to a semantic information unit such as a news story, find other parts of the same or related semantic units

4. allow a user to put in further annotations as additional information related to a semantic information unit like a news item,

5. allow the user to search through the collection of news videos using any prior annotation that the user has made.

Fulfilling the contract required us to develop a prototype *Video Data Management System*, which demonstrated the infrastructural framework for exploring the research issues listed in the proposal. The system has both hardware and software components which together allow the complete management of video data. The fulfillment of the proposal also led to the development of the Virage Video Engine, a complete multi-media engine for indexing and retrieving video based on its content. The Virage Video Engine can process all the three components of video, namely, the image sequence channel, the audio channel and the closed caption channel.

1.1 Organization of the Report

In this report we present the process of designing and developing the video data management system. We also present the technical details of the algorithms used and the infrastructure developed. Section 2 presents the technical objectives of the project and also an overview of the component technologies used in the project. The prototype system included a significant hardware component, the architecture and design of this subsystem is presented in 3. The core content-based indexing and retrieval algorithms are encapsulated in a software engine called the Virage Video Engine (VVE) which is discussed in section 4. Evaluating the system to identify its strengths and weaknesses is an important step in the development of real systems. The evaluation criteria for a video data management system and the evaluation of our prototype using these criteria are presented in section 6. The conclusion of the report is presented in section 7.

2 Objectives and Overview

This section presents the technical objectives of the proposal and an overview of the existing technology,

2.1 Technical Objectives

The technical objectives for the Phase I research were specified as follows:

1. create a set of fully automated video indexes based on in-house and state-of-the-art techniques of shot segmentation, motion segmentation, audio segmentation, key frame extraction and analysis.

2. extract a set of key expressions from the transcribed text of the news, displayed captions and interactively or manually entered annotations where necessary to index the input video stream. An interesting subgoal is to investigate the extent to which video segmentation techniques can be improved by using the parallel set of key expressions extracted from news transcripts

3. develop a visual dictionary (called thesaurus in the proposal) structure to associate related key expressions, related image features and key expressions with image features

4. develop a method to use the visual thesaurus to jointly index images and word expressions develop an abstraction scheme to relate various degrees of abstractions of the news video (such as the headlines, the facts and the details)

5. develop a small set of query processing and browsing schemes to retrieve information at a user-specified level of abstraction using linguistic expressions, image similarity, sequence similarity and combinations of these

6. integrate the techniques developed into the Virage software

The deliverable items for the project are:

1. reports and scientific papers describing design decisions, data structures, feature computation algorithms and their performances, and experimental results for different retrieval strategies.

2. a prototype application containing a video insertion module, a feature computation module, an annotation module and a query module. This prototype would allow users to:

   (a) insert a new news video clip for feature extraction and indexing

   (b) perform interactive annotation through a graphical interface

   (c) perform a fixed set of queries, and browse through the video collection using a simple graphical interface

   (d) choose between image-only, audio-only, language-only and combination indexes for video retrieval and browsing.

During the project period we have created the hardware and software infrastructure to build the news database application, and have a developed prototype system and experimental implementation of algorithms for video analysis, indexing and query in the domain of CNN news.

2.2 An Overview of the Component Technologies

Although the focus of the project is on research in the areas outlined in the previous subsection, engineering a prototype required us to survey the different pieces of technology required. In this section we present a brief overview of this process which involved identifying the exact details of the pieces, determination of the alternatives available and making a decision on what to incorporate in the feasibility prototype.

The process flow for the intended application consists of the following steps:

1. Analog video from the video source (a VHS tape) is digitized into a suitable compressed format. Ideally, the compressed video can be both decompressed and frame accessed for analysis. Also ideally, the compression and decompression of the video occurs at near-real time.

2. Analog audio is separated from the video and digitized to a suitable format. Ideally, the audio is synchronized to the video by by SMTPE time-code or equivalent.

3. Closed caption text is separated from the video and stored as ASCII text. Ideally, the closed caption text is frame synchronized.

4. Time segments of digitized video, audio and text are inserted together as multimedia objects to the Virage System for analysis.

5. Each multimedia block is analyzed by the Virage system to extract

   - shot boundaries and keyframes from image sequence
   - motion properties from image sequence
   - speech, music, silence and other relevant boundaries from audio
   - speaker and other context transitions (like anchorperson to on-location reporter) from video or audio or closed caption text. Ideally, more than one media is used to locate such transitions.
   - textual content from closed caption text
   - textual information from the user to be treated similarly as the textual descriptor acquired from closed caption medium

   Once extracted, the Virage System stores the information as a compact persistent object (feature vector) for future use.

6. The user, through a graphical user interface formulates queries on information extracted from one or more media

7. The Virage System uses the stored feature vectors to find multimedia objects which satisfy the query. The query classes allowed are:

   - Query by Multimedia Value: the system returns all multimedia objects that satisfy a user-specified value condition
   - Query by Multimedia Example: the system ranks all multimedia objects in terms of their similarity to a user-specified multimedia object
   - Navigation: the system plays back the video from a location in the multimedia stream, based on a location finding condition specified by the user
   - Visual Dictionary Query: the system performs a set of multimedia queries, based on a textual query performed by the user

8. The Virage System presents query results through a graphical user interface. Ideally, different query responses will be presented through a different results interface.

Most of our engineering decisions were in the first three of these steps:

Digitized Video Format: We considered AVI, QuickTime, and MPEG-I as potential compressed formats we use as the default representation of digitized video. AVI was not used because it is a stream-only format that does not provide frame access. Although QuickTime allows frame access, it is proprietary format of Apple, Inc. and is available on fewer platforms than

MPEG, which is the ISO standard. It was decided that although at a future date the application should support several formats, for the feasibility study period it is more meaningful to work with a format which can be programmatically manipulated to fit the application. For example, MPEG allows a user to include customized information in the compressed stream for later reuse. One of the tasks for which we needed to use this ability was to store the analog frame numbers of each video segment which was encoded as a stored multimedia object in the Virage System. The encoding enabled to relate the multimedia object to its source analog segment while playing back the original video as the response of a query. We used the Berkeley MPEG encoder and decoder for the purpose of this study.

Digitized Audio Format: Although the intended application needed audio capability, it was decided inappropriate to build in-house expertise with commercial technology available for integration. Our brief survey revealed that very few companies have a feature extraction engine that is easily integrable with third party software. The products that do have such capabilities are mostly speech recognition oriented and do not accept audio from video feeds that contain a significant proportion of non-speech audio content. To be integrated with our proposed work, these products definitely need a preprocessing step to filter out speech. Our preliminary survey also revealed that most of the speech algorithms are not designed to work for speech mixed with background noise, something that is bound to happen for any live video recording. This led us to work with MuscleFish, Inc. an audio database company with content-based retrieval capabilities. It was mutually agreed with MuscleFish, Inc. to use the AIFF format as the representation of digitized audio.

Closed Caption Decoding: It was determined that frame synchronization of closed caption text was an important first step to integrate linguistic information with video. Our survey revealed that none of the vendors who provide closed caption support on audiovisual equipment provide any means to route the text to a computer. Moreover, some stand-alone hardware devices that do allow external decoding of the closed caption information from the video signal provide no frame synchronization. It was also found that most of the video server systems commercially available do not provide frame-synchronized closed caption to the video requester.We therefore decided to have a specialized hardware device built specifically for the project.

3 Video Hardware Architecture

The design of the video hardware system was motivated by the following factors.

Random Access Video: Keeping in mind that the ultimate goal of the project is to be able to search and browse video based on imagery and language analysis, the storage of the video data has to be on a media which supports true random access of video.

Creation of Digital Video Proxies: The hardware system should also support the creation of digital video which will be used for the analysis and indexing of the video stream.

Large Video Volumes: The demonstration required at the end of Phase 2 of the projects requires a system capable of providing content based querying and browsing for at least 12 hours of news video.

Complete Video Input Output Facility: The final demonstration system (at the end of Phase II) should be capable of taking in video from a video tape and putting the results of the search onto a video tape.

Commercial of the Shelf Technology: The final demonstration system should maximize the use of COTS technology.

Keeping these constraints in mind, we have acquired the following hardware for the project:

1. SONY Hi-8 Recorder/Player (SOH-EVO550H) with 10 Hi-8 MPX Pro MP cassettes (2 units).
2. SONY CRV (laserdisc) Recorder/Player (SOH-LVR3000AN) with 10 laser discs (1 unit)
3. SGI Indy-5000 computer (1 unit) with MJPEG video compression board and audio capture board.
4. SONY Trinitron (27 inch) Television Monitor
5. Closed Caption Decoder (1 unit)

3.1 Design Rationale

The rational for each of the individual hardware subsystems are

Hi-8 Tape Systems: The tape systems were acquired to support the complete video input output facility. Since almost all video data originates on tape, The results of searches will also be output onto a tape. The dual deck system is driven by nonlinear editing software which allows the search results to be edited into a suitable format based on the application.

LaserDisc System: This system was chosen for two reasons, it provides random access to video, and can store video at full resolution. This will allows the user to access and view the results of video search with *zero cueing delay* and at full NTSC video resolution. The laser disc also provides the frame accurate access which allows full resolution digitization of the video using generic digitization hardware. The frame accurate access also allows for experiments with digital video tracking while keeping the video on the video disc.

SGI Computer System: The standard software development and testing platforms at Virage Inc. are SGI's. The SGI systems also provide good support tools for manipulation of digital video formats.

TV Monitor: The Sony TV monitor provides the viewing system for the video data. It also provides a built in *closed caption decoding* system against which the results of the external closed caption decoder can be verified.

Closed Caption Decoder: The decoder provides the ability to extract the closed caption information from the television signal and stream it out on to a RS232 serial link as a ASCII character stream. The closed caption decoder also provides *relative time information* (in terms of frame numbers) along with the closed caption data. The time information allows the synchronization of the text and image streams.

3.2 Hardware Architecture Description

The following is a short functional description of the hardware system that is currently being designed and implemented at Virage Inc. as part of the DARPA Phase I grant (see Figure 1). All the initial video data is on video tapes in the Hi-8 or VHS format. The Hi-8 video decks are controlled by software on the SGI system. Video Deck drivers provides all the basic video player control functionality. The video output of the tape system forms the video input to the Laser Disc System. The Laser Disc is also controlled by an RS232 interface which allows the synchronized control of the tape system and the laser disc system. The video data from the tapes is recorded onto the laser discs which are the secondary storage media for the video data.

The audio-video outputs of the laser disc player form the inputs to the video and audio digitization hardware on the SGI. The video signal output is also used as the input to the closed caption decoder. The Laser Disc player is the main data source to the digitization process. The RS232 control software for the laser disc player provides the all the video player controls along with the disc recording control commands.

The video outputs form the laser disc player are used as inputs to the video tape system. The results of a video search are used to control the laser disc player for analog playback. The tape system will be used to record the search results.

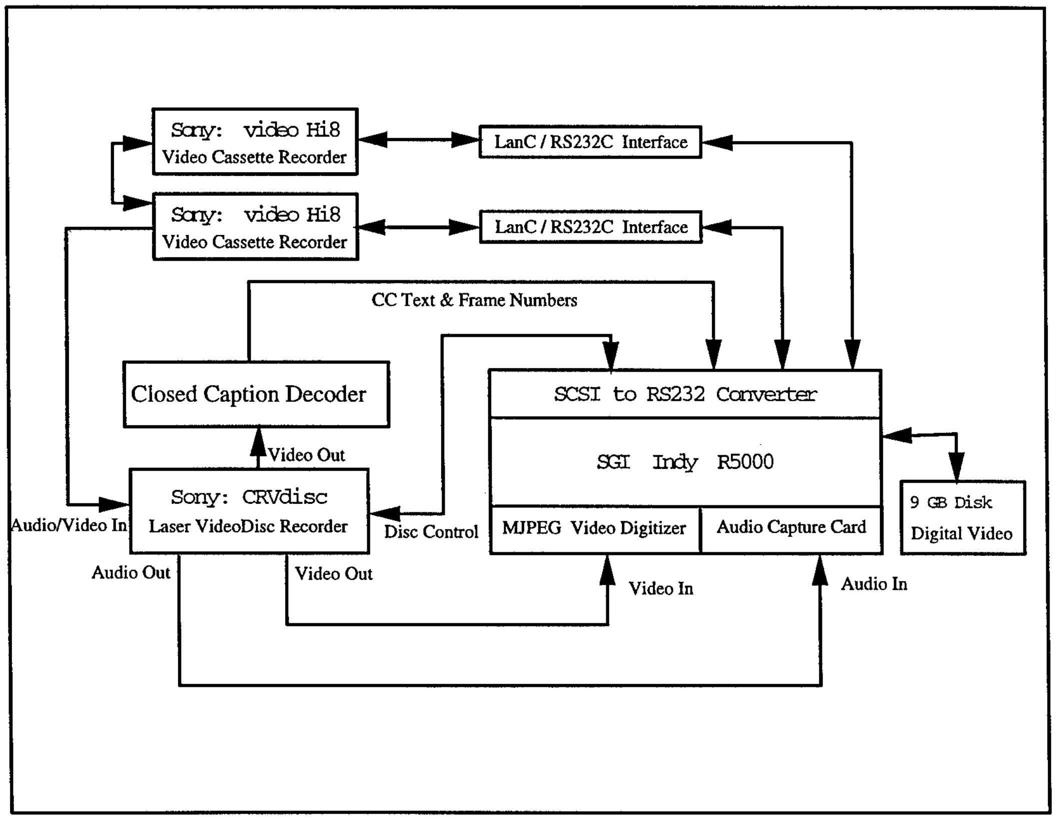


Figure 1: Architecture of Video Hardware Setup

4 The Virage Video Engine

The idea of a video engine is conceptually similar to that of a database engine. The video engine is a tight encapsulation of the core functionality required for automatically processing video streams to generate indices into the data and retrieving data based on these indices. The engine is designed to be independent of the actual backend data store and the application components which provide user interactions. The VVE can be used to build an application which utilizes any database system and application specific graphical user interface. We first present a discussion of the design rationale we used as a guideline. This is followed by a description of the engine architecture followed by a description of the currently implemented video primitives. The video engine has evolved from the Virage Image Engine (VIE) [1] and uses many of the concepts and ideas used in the image engine.

4.1 Design Rationale

The design of the software system is being driven by the following requirements derived from the problem description.

Multi-media Analysis: The problem requires that the imagery, audio and closed captioned text components of the video stream are analyzed in synchrony to compute features. Since the synchronization is not an engine-related task, the audio and the text components would have to be pre-synchronized to image frame by the application software before they are processed by the engine. The engine itself must consider these components to be independently processable by three different component engines. For the imagery and audio component engines *analysis* implies finding index points which divide the input into "segments", finding a representative feature-set for each segment, pack the segment-features into an efficient data structure and return it to the application. For a news video this means identifying and characterizing keyframes and audio transitions. The task of the text component engine is to create a text "feature" that indexes each significant expression to a set of frames where they are likely to have appeared.

Integration of Machine and Manual Annotation: An important task of the application software invoking the engine is to allow an annotator to put more information into the machine-processed videos. This additional information must then be subjected to alphanumeric processing. An example would be to add the name of the newscast with a news segment. A more complex example would be to add a semantic reference to the news segment, potentially relating it to other news pieces. To facilitate the task of integration our design will allow an "update-by-query" strategy available in database management systems. Thus, the user would be able to first issue a query to find similar visual or auditory segments from the database and then add the annotation to correlate them. We think that this design will facilitate the process of grouping video segments into stories, and categorizing stories into story groups.

Multi-media Queries: The multi-media query can potentially use imagery, audio, text, and annotation to set the search conditions. While the individual component engines have the ability to compare feature vectors, the media engine needs to have the ability to parse a multimedia query and dispatch it to the component engines. Similarly, it must assimilate the results returned by individual engines into a consistent response for the user. Ideally, it should also perform some degree of optimization and have the ability to maximize search efficiency, but we shall not address this requirement in this phase of the project.

4.2 Architecture

The video engine is built using a number of components. The framework of the engine allows for these components to be changed by the application developer depending on the application. The two main concepts used in the video engine are

The Media Object: To consider every media (video, audio, text) as an information bearing entity, we devised the media object as an abstraction of a generic "media" data type with uniform behavior, and derived individual media types as its subclasses. The media object provides the necessary isolation between the engine and the specific details of the various formats in which data is typically stored. For example, video can be stored in numerous formats like MPEG, AVI, QuickTime, etc., the media object hides these details from the engine and provides a uniform interface irrespective of the underlying data format.

The Primitive: The primitive is an abstraction of a property computable from any media. For example, for audio the primitive computes properties like means and variances of loudness, pitch, spectral bandwidth, and duration of a sound unit. By design a primitive operates on a specific media type, and contains methods for both extraction of features from the media object and comparison of features. Feature extraction occurs during analysis phase when the media object is inserted into the database. Upon feature extraction, the primitive returns to the application program a compact representation extracted from the data called the *feature vector* [1]. Feature comparison is performed at query time. The result of feature comparison is a *value*, which, depending on the query, either depicts a value returned from the query or a score, indicating how close a query feature vector is to a target feature vector.

The video engine also uses several other concepts like weights, scores and schemas for managing various combinations of primitives and for usefully combining the outputs of these primitives. The reader is referred to Appendix A for a functional description of the Virage Image Engine Architecture. The additional features of the video engine beyond the VIE [1] are:

Multimedia Capability: Unlike the Virage Image Engine, the video engine is designed to accept multiple, potentially asynchronous media streams as a single data object. For the purpose of this project, however, all the three different media, namely image sequences, audio and video have been "frame indexed" to achieve media synchrony. Although it has not been done till date, our multimedia engine is designed to work in conjunction with a video server for any realistic application requiring on-line analysis of continuous video feed. This can be performed with our engine because it can work in a synchronized-clock mode for multiple media streams from an external driving software such as a video server. As part of our Phase II development, and for independent commercial deployment we intend to build application suites with commercially available video servers from vendors like Oracle or Informix.

Time Based Media: Time based media like audio and video have to be treated as media streams rather than as media objects. The key distinction being, that a media stream cannot be processed as a single entity due to its extent where as a media object can. For example, an image can be processed as a single entity, whereas a video is processed one frame at a time and audio is processed in terms of some suitable time buffer (typically 40ms). The VVE provides the necessary support for dealing with temporal media streams of differing data rates. This has significant implications in the way the media analysis function works. It extends the notion of primitives of the Virage Image Engine to allow "multi-pass" primitives. Hence, unlike the Image Engine, a primitive in the Media Engine can implement "backtracking" algorithms. For example, algorithms to detect gradual scene transitions (such as a dissolve operation) have to pass through the media stream once to collect potential candidates for dissolve boundary. It "looks ahead" downstream in search of a point in the media stream where the dissolve has already passed. Once such a point is detected, it revisits the media stream to locate which of the potential dissolve transition points marked in the earlier pass are the true dissolve boundaries. Use of the time-based media also allows a primitive to time-tag keyframes. This helps to use a temporal constraint in the comparison of two sets of key frames in the process of finding visually similar videos.

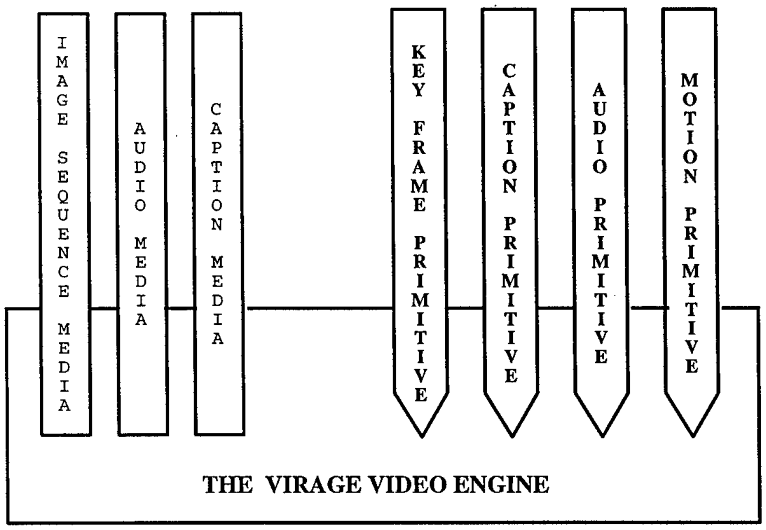


Figure 2: The Virage Video Engine

4.3 Image Sequence Primitives

The image sequence in video is the highest resolution channel. It provides the most detailed information about the events that are being communicated or captured on the video. The image sequence in a video can be indexed along two dimensions namely, visual and temporal. The VVE provides primitives for extracting indexes along both these dimensions. The Virage KeyFrame Primitive extracts visually significant keyframes to represent the visual content of the image sequence. The Motion Primitive extracts indexes which describe the motion content of the image sequences.

4.3.1 The KeyFrame Primitive

The function of the keyframe primitive is illustrated in figure 3 (left). Given any image sequence, the keyframing primitive extracts frames from the image sequence, which conform to the following definition of an *optimal key frame set.*

Definition: Given the image sequence component **I** of a video **V**, the keyframe set **K**, is defined as the minimal number of frames sampled from the video **V** which provides an *adequate representation* of the visual content of the video.

The term *adequate representation* is application dependent. For example, in the context of a news video retrieval and browsing environment, it may be necessary to extract both shot boundaries and visually significant frames (keyframes), whereas in a film production environment, where most of the videos are raw unedited dailies [8], just extracting the shot boundaries may suffice. On the other hand, for non-produced video as may be created by mounting a video camera on a UAV, only keyframes and no shot boundaries will be meaningful.

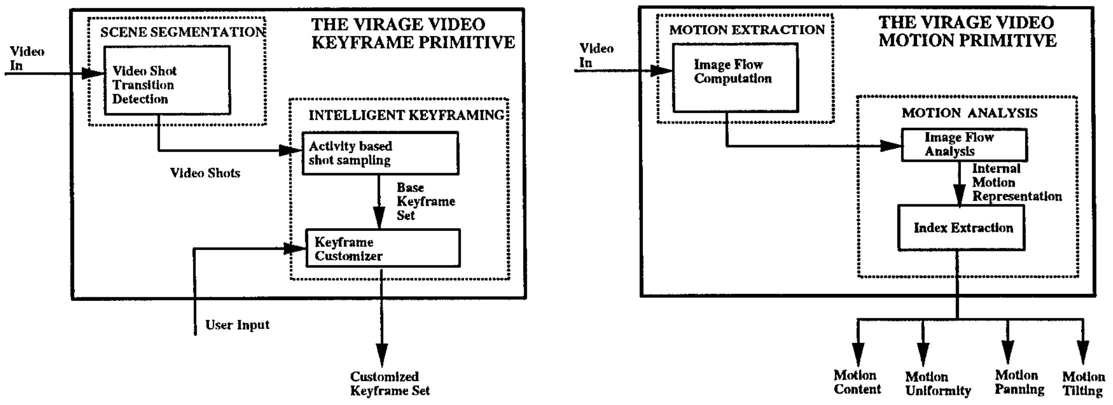


Figure 3: **Left:** KeyFrame Primitive **Right:** Motion Primitive

The Virage keyframe primitive (figure 3 (right)), is designed to support this application specific nature of the keyframe set. The keyframe primitive first determines shot boundaries checking for both abrupt and gradual scene transitions like dissolves, fades, and wipes. Within each shot it uses a computationally efficient proprietary adaptive sampling algorithm, to extract an initial set of frames from the video called the *base keyframe set.* This set of keyframes is then processed using the image primitive set supported by the VIE. The resulting set of frames is called the *customized keyframe set*, which can be viewed as a **summary of a unit of video footage**. Following the parlance of the entertainment industry, we call this summary an *Inverse Storyboard.* Since different applications have different requirements for a summary video, the keyframe primitive has been designed to provide the user some "knobs" to control and alter the customization process. The customization process can be tuned by five parameters: the weights of the four basic image primitives supported by the *VIE* and a threshold on the overall distance between successive base keyframes. Figure 4 shows the effect of tuning the distance threshold to control the process of customization. It can be observed from the screenshot that the similarity setting which produced fewer keyframes may sometimes filter out keyframes which are visually significant. In our experiments we have determined that in most cases adjustment of some of the five controls produces a semantically acceptable summary.

The keyframe extraction process is discussed below. Each of the steps used is listed in order and the details of the algorithm used in each step are presented. Given any video object **V** it is processed by the following steps to extract keyframes.

Segmentation: This is the process of decomposing the video **V** into shots based on the location of artificial scene transitions like cuts, dissolves and fades. A shot is defined as a continuous camera take [8]. A proprietary algorithm is used for the process of detecting shot boundaries. The keysteps used in this algorithm are listed below.

1. Given any image V_t in the video, compute a set of measurements based on some local temporal neighborhood of t, namely $t - n, t + n$.
2. The measurements set is typically a set of local image difference measures, a set of global image difference metrics, a set of tuned measures for detecting special effects used in soft transitions.
3. Each of these measures is thresholded into a three level measurement.
4. The thresholded measurements are combined using a decision table, to decide weather a particular frame is a transition frame.
5. The last process in the shot segmentation is a temporal filtering step which groups transitions into transition segments based on temporal locality.

Fast Adaptive Sampling: The adaptive sampling process is tuned to apply to video shots (i.e. video segments which do not contain any artificial edit effects). The key idea is that, from each shot we should be able to extract a set of frames based on the activity in the shot. These set of frames can then be run through a more expensive process of visual similarity based comparison.

The adaptive sampling is a global process, in that it examines the entire shot before deciding on the locations of the keyframes within the shot. The steps involved in the adaptive sampling process are listed.

1. The total number of samples extracted is a user definable quantity.
2. The distance between consecutive frames within the shot are computed and stored. The distance can be any type of inter frame difference. These distances are typically available from the shot segmentation process.
3. The shot is then segmented into sub-shots bases on regions of uniform activity.
4. The required number of samples is distributed among the segments based on the ratio of the activity in each sub-shot to the total activity in the shot.

Visual Feature Extraction: Once we have a set of adaptively sampled frames from each shot called the base keyframe set. The visual properties of the base keyframe set is extracted using the standard Virage image engine [1] and the Virage set of primitives.

Visual Filtering: Using the visual properties and the Virage similarity comparison, the base key frame set is filtered into smaller sets of keyframes. This filtering process uses a user defined threshold value which controls the number of keyframes chosen and a user defined weight set which controls the visual relationship between keyframes.

4.3.2 The Motion Primitive

The Virage motion primitive extracts indexes into the image stream based on the motion content of the image stream. Figure 3 (right) illustrates the high level structure of the Virage motion primitive. The motion primitive operates on the image sequence and extracts from it the image flow [7, 5]. The image flow is a representation of the pixel motion between consecutive images in the image sequence. The flow information is transformed into an internal intermediate representation which is the basis for extracting the motion indexing information. The index extraction process indexes the clip based on several simple properties of the motion with the video. The dimensions of the motion index are:

Motion Content: This is a measure of the total amount of motion within a given video. From a user's perspective this translates to the *action content* of the video. For example, a *talking head video* will have a very small motion content measure where as a *violent explosion* or a *car crash* will typically have a high value.

Motion Uniformity: This is a measure of the smoothness of the motion within a video as a function of time. For example, a smooth panning shot, or a tracking shot [8] will have a high value of motion smoothness where as a video which has a staggered pan will have a low value.

Motion Panning: As indicated by the name, this dimension captures the panning motion (left to right, or right to left motion of the camera). A smooth pan shot scores high as compared to a zoom.

Motion Tilting: This is a measure of the vertical motion component of the motion within a video sequence. Panning shots will have a low value as compared to videos which have a large amount of vertical motion.

Since a video object is first split into shots and transitions, the motion primitive operates only on the shot segments and not, for example, on dissolves. In the domain of news the motion primitive works efficiently on most scenes. However, for some outdoor scenes which have a combination of camera motion, multiple object motions and camera jitter, image flow computation becomes uninformative. In our future work, we shall explore more motion processing techniques.

The detailed working of the Virage motion primitive is discussed below.

Flow Extraction: The image flow between a pair of frames in the video shot is extracted using a proprietary image matching algorithm. The algorithm is fast and is tuned to operate on realistic video sequences. No assumptions are made about the motion except the maximum interframe motion.

Flow Histograming: The interframe flow is processed and a histogram representation of the flow is extracted. This is the intermediate representation used by the motion primitive.

Motion Labeling: The flow histogram is used as the basis for ranking the videos based on the different types of video queries.

4.4 The Audio Primitive

The audio track carries a significant amount of information in any produced video. The Virage audio primitive is designed to process the audio component and segment the audio information based on significant changes. The actual semantic information in the audio track is typically contained in the speech portion of the audio track and requires sophisticated speech recognition technology [3] for analysis. However from a video indexing and retrieval perspective coarser level changes in the audio track provide very valuable index points into the temporal video stream. The Virage audio primitive (Figure 5 (left)) was developed based on the technology of MuscleFish Inc. [12] an audio search and retrieval company. We adapted their audio feature extraction capabilities to work in the context of the Virage media object and primitive. The resultant audio primitive accepts an audio stream and computes "audio keyframes" defined as segments over which the frequency bandwidth, loudness, and brightness do not vary significantly. These segments are processed to compute audio statistics (see [12] for details) for the segments in a manner similar to the image keyframes. Unlike the image sequence primitives, the audio primitive allows the user to define exemplar-based classes, leading to the following query classes:

1. Find an audio segment similar to a given query segment
2. Find in a given audio stream instances of audio segments belonging to a given predefined class
3. Find instances of the audio segment that transitions from segment class A to segment class B the following types of transitions within the audio stream.

For the news application we used these basic capabilities to determine the following audio segments:

Silence Zones: These are points in the audio stream which have minimal or no audio activity associated with them. Depending on the domain of the video, these index points are indicators of important events within the video stream.

Speaker Changes: This is referred to as speaker segmentation in speech processing literature [11]. In case of news video, speaker changes indicate important events like hand off from reporter to anchor person and vice versa, question, answer points during an interview, etc.

Speech-Music Changes: These are index points into video where music ends and speech begins or vice versa. These transitions again indicate important points in the video from the information content perspective.

Although the audio primitive functions reasonably for several cases we have experimented with, it requires significant tuning in terms of threshold and time-window adjustments to fit into any specific application. Also, the exemplar-based classification framework used by the MuscleFish feature extraction process requires a large training set to distinguish between speakers, and sometimes between music and speech. We believe we have to devise a two-pass algorithm to improve the audio keyframing task by first finding oversegmented audio keyframes and then using a different window and threshold set to prune the segments. In addition to such tuning, the level of indexing provided by the current Virage audio primitive can be enhanced along several directions. One of the directions is to provide primitives which can perform similarity based searching on the audio track. This will allow for the audio track to be indexed based *key sounds* which occur in particular applications [2]. Further primitives can provide even more sophisticated audio and speech processing components like *word spotting* [10]. Primitives based on speech recognition can be added as the technology matures and stabilizes [6].

4.5 The Closed Caption Primitive

The closed caption information when available encodes most of the speech component of audio as ASCII text. Thus having a closed captioned video is equivalent to having a high accuracy speech recognition system operating on the speech portions of the audio track. The closed caption signal [9] is an encoded data signal in the unblanked portion of Line 21, Field 1 of the NTSC video signal. The composite data signal has a clock run in signal, start bit and 16 bits of data. This results in a data stream rate of approximately 480 baud. The encoded data contains information for display formatting, language selection and the alpha numeric character information. The closed caption information is decoded from the analog video signal using a special hardware decoder. The decoder filters out the undesirable information such as display formats and outputs the ASCII text from the captioning and some frame synchronization information. This ASCII text is stored and becomes the input to the caption primitive.

The closed caption contains both alphanumeric and non-alphanumeric information, the latter being used to designate transitions – including speaker transitions, topic transitions, and voice transfer from an anchorperson to an on-location reporter. The caption primitive first preprocesses the input stream to remove punctuations and non-informative symbols, and then parses it into lexical tokens. The transition tokens are are accumulated into a separate feature. The words are filtered through a stoplist, and are indexed into a variant of a balanced binary tree. This data structure acts as our implementation of an inverted file, and also stores word frequencies and points of word occurrence (frame numbers) in the input stream. We plan to adopt a more robust implementation of the text index at a future time. It may be noted that the closed caption index is used both as a search tool and a navigational aid. As a search tool, it is used to answer a query like "Which video objects have the word **Clinton** mentioned in it?" As a navigational aid it is used to jump to "the next (previous) topic (or speaker)" within the scope of a news broadcast. A difficulty encountered in the implementation of the closed caption primitive is that despite an explicit frame synchronization performed at the hardware level, the text stream always lags behind the actual frame of the spoken word. In general, retrofitting the caption boundaries to speech boundaries needs to synchronize the audio segments to the known speaker boundaries. We have not looked into efficient techniques for the audio-text alignment in the course of this project, but plan to implement such a mechanism in the next phase.

5 Building The News Retrieval Application

A complete video database management system using the VVE for content based video retrieval needs a number of components besides the video engine. Figure 6 shows a high level view of the News Retrieval application. The interactions between different modules are discussed by looking at the insertion and retrieval processes.

Video Insertion: In the application prototype created for the project video data is collected and digitized in a separate offline process. Therefore the task of insertion is to read in the predigitized streams of video and analyze it to compute the video feature vector. As discussed before, the design decision in this procedure is to determine what constitutes a storable chunk of information for insertion. We determined that storing an entire news session (minus the commercial breaks) as a single insertable unit is makes the storage object too long for loading, querying and manipulation. Storing the information in terms of keyframe-segmented or shot-segmented units, made the storage unit variable in size and also potentially too long to manipulate. As a first pass the we decided to partition the video into about 30 second clips, partitioned at the next higher caption boundary. This decision is far from optimal in several situations, and we are exploring into finding an improved set of criteria to determine a storage unit of video. Once the input video is partitioned into these storage units, the *collection management module* keeps a record of all information required to access the feature vector, pointers to the original video data, and video thumbnails (vicons). Currently we have not used any other information about the video other than the primitives described earlier. At a later time any collateral video data like video name, production date, etc., will be stored in the *metadata server*, through the metadata management module. In our plan toward commercialization of the technology developed through the project, the metadata server is envisioned to be an external commercial DBMS. In Phase II of the project we shall determine how to create a uniform interface to communicate with a relational or object-oriented database for insertion and querying of videos. It may be noted that we separate the collection management module from the metadata management module, because in an actual application, the metadata management module will possibly reside at the server end while the collection management module will be at the client side and communicate with the metadata server. In a similar vein, the *data management module* is the client side of the video server that knows which video server files need to be analyzed and/or played back, and in what order.

Video Retrieval: During the query process, the *query processor module* analyses the user query and suitably configures the VVE. The *collection management module* then accesses the metadata for objects in the collection and calls the video engine with the appropriate feature vectors.The video engine compares the feature vectors and returns the results of the query. The results of a query are managed by the *result processor module* and presented to the user through the *graphical user interface*. It is necessary to point out that just like a commercial database front-end needs different forms to formulate different queries, we need very different interfaces both to formulate different user queries in the most expressive manner, and to present the results in intuitive ways. Although we have not emphasized on user interfaces for this project, we have created the basic the software infrastructure for management of multiple query classes. The details of currently handled query classes are described in the following subsections.

5.1 Multi-modal Access to Video Information

The distribution of information across the different channels in the video signal depends very heavily on the application domain from which the video is drawn [4]. For example, typically news video has the highest information content in the audio channel, followed by the image sequence channel. Whenever available the close caption channel contains the same information as the speech component of audio. Table 1 shows the relative distribution of information in the different channels for different types of video. In general produced video has high information content in the audio channel and medium information content in the video channel. Depending on the availability the information in the caption channel can be either high or none. In video sources like security camera videos, the information content will be high in the image channel and medium in the audio channel. An application like aerial survey on the other had capture information only in the visual channel. Thus depending on the application, the indexing strategy for video changes dramatically. The Virage video engine addresses this problem by providing primitives which encapsulate different types of indexing functionality. The engine can be reconfigured using a different set of primitives based on the application.

Video Source	Image Channel	Audio Channel	Caption Channel
Produced	Medium	High	High/None
Security	High	Medium	None
Aerial Survey	High	None	None

Table 1: Information Distribution across channels

Based on interactions with potential users of video databases we have identified *multi-modal content cueing* as one of the central problems in video data retrieval. The following is a definition of the video content cueing problem.

Video Content Cueing: Definition: [8] Given a video **V** of time duration **T** the task of marking (digitally) the points of potential interest **C** $= \{ t_1, t_2, t_3....t_n\}$ based on some user specified *content description or query.*

The content queries that can be supported are dependent on the primitives that are supported by the VVE. Similar to the image engine [1] the video engine also supports user defined primitives. The set of default primitives for the VVE have been discussed in section 5.1. The following is a discussion of the types of queries that can be issued based on the *default video primitive set* namely, keyframe, motion, caption and audio.

5.2 Image Sequence Queries

These are queries which retrieve video based solely on the image sequence content of the video. Such queries are interesting mainly in application domains where the video content is primarily visual. The types of queries supported are the *visual similarity queries* and the *motion ranking queries.*

5.2.1 Visual Similarity Queries

These queries rank all the objects in video database based on its similarity to the query object. The query object can be either an image or another video.

Video Queries: Here the query object is the video. Such video queries can be used to search a collection of videos for *clip containment*, i.e. to find other videos that contain, the query clip or other video objects in the database that are contained within the query video clip.

Image Queries: Here the query object is an image. This query basically answers the question *Find me all videos which contain images similar to the query image.* Here image similarity can be defined by the user in terms of the standard Virage image primitives [1] namely, color, texture, structure and composition.

Visual Dictionary Queries: The query binds a text term (e.g. Bill Clinton) to an artificial video object created by composing a number of image examples describing the term (in this case different images of Clinton as normally shown in the news). The Visual Dictionary query performs a "nearest video" query on the database with the artificial video as the query object. As a result, the system ranked the stored video objects in terms of their closeness to *any* of the constituent images of the "dictionary". Effectively, this implementation results in a "society of disjunctive models" query against the image database.

5.2.2 Motion Ranking Queries

These queries rank a collection of videos based on the motion properties of choice. For example, using the *motion content* aspect, results in each video object being assigned a number between 0 and 100 based on the total amount of motion in the video object. Using the motion content ranking queries on different on a database of generic video clips, and on a database of news video we experimented with, have yielded the following interesting results.

Object Versus Camera Motion Clustering: Video Objects with no motion or very little motion have low values and objects with significant camera and object motion have high values on the motion content index. Videos with only object motion rank in the middle.

Commercial Clustering: Commercial on cable television feed typically get a high measure on this index since they tend to have a significant amount of motion within a short time span.

Talking Head Clustering: Talking head segments in news videos tend to cluster with low values of the index.

5.3 Closed Caption Queries

In our current implementation the closed caption queries can only take single words to locate storage units which contains the word. As mentioned, navigation by topic and reporter transition is also allowed. However, we have not implemented a full text retrieval system on closed-caption information at this stage. We believe it is a better engineering solution to use our closed caption primitive as a uniform interface to communicate with an industrial strength text retrieval system.

5.4 Combined Queries

An application which uses the VVE should combine the results of various queries and present a combined multi-modal query to the user. Using multimodal queries results in a reduced set of video clips and reduced number of cue points with in each video object. This results in a reduction of the total information returned to the user in response to a combined query. An typical multimodal video query is presented below.

```
Find Videos with
Caption    = <Bill Clinton >
  (caption subject index  :  caption primitive)
Visual     = <Clinton Image>
  (visual image query     :  keyframe primitive)
Motion     = < High Action >
  (motion content query   :  motion primitive)
Audio      = < Speech      >
  (music speech transition:  audio primitive)
```

We are currently experimenting to determine how to use the video primitives to generate scores for each of the video objects in the database and use these distances to select portions of the database.

5.5 Visualizing Video Query Results

One of the challenges of dealing with a multi-modal data like video is the design of an intuitive Graphical User Interface. Virage has prototyped several graphical user interfaces for video database application. Figure 7 is graphical user interface for a prototype video database built using the VVE. The Figure shows a results window which displays several *vicons*. Vicons are graphical representations of video constructed by sampling frames out of the original video and compositing them into a single image. The Figure also shows the query window which allows the user to switch between the different distinct query modalities and also allows the user to query based on a particular type of index within each modality.

6 System Evaluation

This section provides an evaluation of our prototype *Video Data Management System*. Our purpose is to first establish hard requirements for a real-world application, estimate how far our system has reached in meeting these requirements, and identify areas that need further development. With this broad goal, Section 6.1 presents the requirements for a complete video data management system as a whole. Section 6.2 lists out the various components used and the criteria for evaluating each of these individual components. Section 6.3 presents the evaluation of the existing system against each of these criteria.

6.1 System Evaluation Criteria

The purpose of the video data management system is to provide the user with, a set of procedures and tools for managing and accessing large quantities of video data. The set of criteria presented here attempt to quantify the success of a system in achieving this goal.

Meta Data Scalability: As a video gets more complex, or the length of a video increases, the metadata computed from the video also grows. Metadata scalability refers to the system's ability to cope with this growth.

Raw Video Scalability: This deals with access of the physical video files for browsing and searching. As the size of video collection grows, every search operation results in the potential access and manipulation of increasingly greater number of voluminous video files. A real system needs to offer methods to minimize unnecessary access to the raw physical data.

6.2 Component Evaluation Criteria

This subsection presents a set of evaluation criteria for each individual component used in building the *Video Data Management System*. The evaluation criteria are qualitative for many of the subcomponents and are better specifiable for others. As the system development progresses, the evaluation criteria will become more concrete and quantitative.

Hardware Subsystem: The purpose of the hardware subsystem is to provide an easy, usable and economical way to convert video data from a standard VHS video tape to a digital format which can reside on a computer disk. The hardware subsystem should also provide an easy way to access the original media. The evaluation criteria for the hardware subsystem are discussed below.

1. **Speed of Conversion:** This is the time it takes to convert data from a VHS tape to digital compressed video.
2. **Quality of Digital Video:** The quality of digital conversion can be measured in terms of the resolution, both spatial and temporal, of the digital video as compared to its analog counterpart.
3. **Access to Original Media:** This is a measure of the accessibility of the original media based on the digital proxies that are returned from the search process.
4. **System Cost:** The cost of the entire system including the equipment and the media cost.

Software Subsystem: The software subsystem is responsible for providing the user with the content based video access facility. The following is a list of key components of the software subsystem and the evaluation criteria adopted for each of these components. There are several other software components which provide auxiliary facilities like, video insertion, manipulation, basic editing functionality, etc., which are not discussed here.

Video Engine: Analysis and Comparison Component: This provides the core functionality for content based search and retrieval. This component uses the media files supplied by the user and operates on it using the the standard *Virage video primitive set* namely, the keyframe, motion, audio and caption primitives. This evaluation measures the representational adequacy of existing primitives and identifies new primitives for unrepresented dimensions.

Multimedia Integration: Given that video is has several different synchronized tracks of information, there needs to be techniques which will use the synergy between the different media components in both the analysis and the comparison during queries. The two aspects for evaluation here are

1. The cooperation of the different primitives during analysis time to improve the fidelity of the extracted representation.
2. The use of queries which use multiple media to perform more effective video search and retrieval.

Graphical User Interface: Video presents a challenging task for designing graphical user interfaces due to its *spatio-temporal* and *multi-modal* nature. The following are some of the criteria for evaluating video graphical user interfaces.

1. **Visual Representation:** These primarily relate to how well the graphical user interface presents the *visual aspects of video*. Ideally the iconic representation of a video would allow the user to easily visualize its information content. Such content consists of properties such as visual appearance, length of the video, busy-ness of the video in terms of its component scenes, busy-ness in terms of motion, etc.
2. **Audio Representation:** Representing the audio content of a video through a graphical user interface is an extremely challenging problem.
3. **Auxiliary Data Representation:** Representing other temporal data like closed captioning, user annotation, etc.
4. **Query Formulation:** The graphical user interface must also support the formulation of user queries. GUI's for specifying motion direction and motion magnitude present challenging problems.

6.3 Evaluation

Here we list each of the evaluation criteria discussed above and provide an evaluation of our current video data management system against these criteria.

System Evaluation: Meta Data Scalability: Given a video object that is represented by **n** keyframes and another object represented by **m** keyframes, the comparison time is **O**(m x n). For a database with **N** video objects the total search time will be **O**(N x n x m).

The prototype system we have built has been tested for video databases of 30 minutes. The searching time for such small collections is almost realtime and hence the system can perform without any form of *indexing*. But as the video collections become more realistic in size (more than 5 hours), the searching time without some form of indexing will become unacceptable to the user. *Video Indexing* is one of the key problems that needs to be addressed before content based video searching becomes a commercial reality.

To develop a video database system of commercial capacity will require that the metadata be stored in a suitable *database* with a well designed indexing scheme. There are several issues here regarding the choice of databases (object oriented vs relational) and design of suitable indexing schemes which needs to be addressed.

Data Access Scalability: The prototype system uses the file system for storing the video data. The data is low resolution (120 x 160, 30fps) mpeg1 proxies. The typical length of these clips is lesser than 5 minutes. Thus the problem of accessing, retrieving and playing these video clips from the file system has not posed any significant problems.

A typical industrial strength video database for example a *news video archive* will have 100's to 1000's of hours of video in a collection. And the resolution of the data will typically be (480 x 640 x 30fps). This will require the use of industrial strength *Video Servers* to store the video data.

Video servers that are currently available from commercial vendors are typically geared towards the *video on demand* market. They are tailored for complete video object retrieval with a VCR paradigm for viewing the video. There are several issues to be explored with respect to how such video servers can be adapted to the browsing and partial viewing situations that are encountered during the content based retrieval situations in a video data management system.

Hardware Subsystem: The prototype hardware system we have built is very good for an experimental video database system. It is been primarily designed to support the process of developing video processing and manipulation algorithms.

Speed of Conversion: The process of converting data from a VHS video tape to digital compressed video is currently an offline process. The steps involved in this conversion are

1. Record the VHS tape onto a CVR laser disc.
2. Play the CVR disc under remote RS232 computer control from the host computer.
3. Set the laser disc player into the frame advance mode.
4. Use the digitizer card and software to grab the current frame and advance the disc to the next frame of video.
5. Once the set of frames for the segment of interest have been digitized, use the software compression algorithm to generate the mpeg 1 file.

This entire process typically will take about 3 to 4 times the length of the VHS video to complete. An industrial capacity video data management system will require an input system which can generate the compressed digital video in realtime and store it on disk. There are several such commercial systems. We need to explore this space of video compression hardware and the associated integration issues.

Quality of Digital Video: The prototype system can digitize video at full NTSC resolution (640x480) at 30fps.

Access to Original Media: This is one of the significant advantages of the prototype system. Since the original data is stored on a video disk. We have complete computer controlled remote random access to the original video media. This is a very useful for browsing, the video at full resolution on a commercial television monitor.

System Cost: The cost of the prototype system is lesser than the cost of the high end mpeg digitization hardware. However there are several low cost options available on the PC platform. The performance and quality of these options are to be explored.

Further Development: The further development of the hardware subsystem will be aimed toward making the digitization process a realtime process. This also implies the need for a large capacity online storage facility for storing the full resolution video data along with the software infrastructure for managing the video data collection.

Software Subsystem: Video Engine: Analysis: The scope of this evaluation will include the current set of primitives and the performance of these primitives on the current data set.

Evaluation of current primitives: The evaluation will focus on

Keyframe Primitive: The key framing algorithm in the current prototype uses a two pass approach. The first pass detects shot boundaries and extracts an over sampled set of frames from each shot. The second pass uses visual similarity based filtering to *filter out visually redundant frames* generated by the first pass. The algorithm provides the user the freedom of defining visual similarity in terms of the thresholds and weight sets used in the visual filtering. The approach is completely domain independent.

There are several fronts on which the performance of the keyframe primitive can be improved. These include, the tuning of the adaptive sampling algorithm and designing methods by which the user can define the objects and events of interest in a particular domain.

Motion Primitive: The motion primitive at this time extracts coarse motion measurements from the video sequence. These measurements are used to rank the videos in terms of motion properties like, motion amount, motion smoothness, panning and tilting.

Experiments on databases of commercially produced video have indicated that the motion amount provides an interesting way of ranking videos. However, on the test databases, the other queries of motion smoothness, panning and tilting did not provide very meaningful results. This is mainly due to the nature of the data, which mainly have sequences which combine panning, tilting and zooming.

Further experiments need to be conducted on identifying suitable domains of video where these attributes provide meaningful results. We also need to explore the types of attributes that can be used for general commercial video broadcasts.

Audio Primitive: The audio primitive is designed to segment audio stream into segments based on criteria like, speaker transition, music to speech transitions, silence zones, etc. These features provide a very low level index into the video based on the audio samples.

Further developments in audio primitives can address the problems of audio keyframing, keyword spotting, speaker identification and gradually evolve towards supporting complete speech to text conversion.

Caption Primitive: The caption primitive provides the basic functionality of searching through video using the associated captioning. The search capabilities can be improved to a great extent by using commercially available *text search* engines. The integration with such engines will have to preserve the temporal tagging of the text data.

Further Developments: There are several other new primitives which can be used to index and retrieve the video, some of are listed below.

Object Track Primitive: This represents the location and motion of objects within a given video clip.

Audio Keyword Primitive: This primitive indexes the video based on keywords recognized from the audio track.

Key Event Primitive: This primitive indexes the video be detecting user defined key events in the image stream of the video.

Multimedia Integration: The video engine used in the prototype is a true multimedia analysis and comparison engine. The media type is completely transparent to the engine, as it uses a standard media encapsulation for all types of media. Evaluation of the mutlimedia capability can be performed based on the following criteria.

Cooperation during Analysis: The prototype does not utilize the synergy that exists between the three synchronized media streams namely image sequences, audio and captions. For example, certain types of transitions in the audio in combination with corresponding scene transitions in video can be used to trigger very specific domain dependent primitives. Further development of the multimedia engine will examine the synergy and cross coupling between different media primitives.

Multi-modal Queries: Our prototype system can repond to queries in each of the independent media. The system does not have any facilities for combining the query results from the different media. Further developments along these lines can include a multi-media query optimizer which can launch a multi-modal query in response to a high level user query, combine and present a unified result to the user.

Graphical User Interface: The design of graphical user interface for a multi-modal temporal media like video is a very complex problem. There challenges include the presentation of time on a spatial graphical interface, presentation of a non visual media like audio and the presentation of the synchronization between the media.

Visual Representation: The prototype system, uses a simple **vicon** which is composite image generated by stacking together the different a fixed number of samples taken from the beginning of each video object. There are several different representations that need to be explored like, the use of active vicons, where the use controls the appearance of the icon using the mouse, making vicons using only keyframes, making icons using animated gifs of the keyframes, etc.

Audio Representation: The prototype built has not attempted to represent audio. There are several different possibilities from representing the waveform of the audio as an icon along with the vicon, representing the type of audio in a video clip using, a coloring scheme for different types of audio and playing back audio at higher rates.

Auxiliary Data Representation: This is an issue that has not been addressed in the current system. There can potentially be different types of media associated with a video stream like caption text, user annotation, graphics, etc. Further system development will address these issues.

The evaluation presented in this section has discussed several criteria for evaluating video data management systems. The current prototype system has been evaluated against these criteria and the short commings and strengths of the system have been discussed.

7 Conclusions

We conclude this report with a brief recapitulation of our goals, major accomplishments, self evaluation and future targets.

The primary purpose of this project was to determine how feasible it is to create a real-life video information retrieval system which will meaningfully use both the visual and the linguistic channels of information. We included an audio component as a third source of information and set ourselves to devise and implement techniques of using all three media together to analyze a generic video stream. The objective of the project was to develop an enabling technology which provides a user with the means to (semi-automatically) annotate, search for information and intelligently browse through long videos.

Aimed at this goal, our foremost achievement was to create a complete hardware and software infrastructure where a user can put in an analog video tape to the system, and the system will convert it to a queriable and browsable digital video collection. The most important component of this infrastructure was the design of the Virage Video Engine with truly multimedia analysis and retrieval capability. Secondly, as part of our video analysis toolset, we developed a potentially commercializable video summarization software module called the "inverse story board". This component used our activity-adaptive two-pass keyframe extraction technique. Thirdly, we created a motion analysis suite which partially analyzes camera and object motion properties in a scene. When fully developed, this suite is likely to have demand in motion pictures and video editing industry, as well as in object tracking and surveillance tasks. Fourth, we successfully customized and integrated two commercial technologies into the fold of the Virage Video Engine, viz. the MuscleFish Engine for audio analysis, and a specially designed closed caption decoder for frame-based caption access. With these components built into the Virage Video Engine, we have demonstrated a prototype system having a limited multimedia query and navigation. Even this small-scale demonstration prototype has generated significant interest among video database providers and possible future users such as in the broadcasting industry.

Equally important as all our achievements in this project is the clarity of understanding we have gained into the actual engineering requirements of the video analysis, storage and retrieval problem. We have evaluated ourselves against very hard, realistic criteria that must be met with, for our efforts to evolve into an "industrial strength" product for the defense and commercial markets. As detailed in Section 6.3 we need further development to make our algorithms and system design more scalable, have more efficient and domain specific video analysis and comparison tools, develop methods to explicitly incorporate user-requirement profiles into our analysis and retrieval framework, and develop more effective cross-media indexing and query facility. We will focus on these areas of research in Phase II of the project.

References

[1] J Bach, C Fuller, A Gupta, A Hampapur, B Horowitz, R Humphrey, R Jain, and C Shu. The virage image search engine: An open framework for image management. In *Storage and Retrieval for Still Image and Video Databases 4*, pages 19–28. The International Society for Optical Engineering, Feb 1996.

[2] T Blum, D Keislar, J Wheaton, and E Wold. Audio analysis for content based retrieval. Technical report, Muscle Fish, 2550 Ninth Street, Suite 207B, Berkeley, CA 94710, 1996.

[3] J S Denton and C R Taylor. Final report on speech recognition research: December 1984 to april 1990. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1990.

[4] Arun Hampapur. *Designing Video Data Management Systems*. PhD thesis, The University of Michigan, 1994.

[5] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[6] X Huang, Fileno Alleva, H Hon, Hwang, and R Rosenfeld. The sphinx-11 speech recognition system: An overview. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1992.

[7] Ramesh Jain, Rangachar Kasturi, and Brian G Schunck. *Introduction to Machine Vision*. McGraw Hill, 1995.

[8] Ira Konigsberg. *The Complete Film Dictionary*. Penguin Books, 1989.

[9] John Lentz, David Sillman, Henry Thedick, and Evans Wetmore. Television captioning for the deaf signal and display specifications. Technical report, PBS Engineering and Technical Operations, Public Broadcasting Service, 475 L'Enfant Plaza, S.W, Washington D.C, 20024, May 1980.

[10] Wilcox and Bush. Hmm-based word spotting for voice editing and indexing. In *Proceedings of the Second European Conference on Speech Communication and Technology*, September 1991.

[11] Wilcox, Chen, Kimber, and Balasubramanian. Segmentation of speech using speaker identification. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing*, April 1994.

[12] Erling Wold, Thom Blum, Douglas Keislar, and James Wheaton. Content-based classification, search and retrieval of audio. *IEEE Multimedia Magazine*, pages 27–36, Fall Issue 1996.

8 Appendix A

A Functional Description of the Virage Image Engine

8.1 Data types

A content-based information retrieval system creates an abstraction of the raw information in the form of features, and then operates only at the level of the abstracted information. For visual information, features may belong to five abstract data types: values, distributions, indexed values, indexed distributions, and graphs. A value is, in the general case, a set of vectors that may represent some global property of the image. The global color of an image, for example, can be a vector of RGB values, while the dominant colors of an image can be defined as the set of k most frequent RGB vectors in an image. A distribution, such as a color histogram is typically defined on an n-dimensional space which has been partitioned into b buckets. Thus, it is a b-dimensional vector. An indexed value is a value local to a region of an image or a time point in a video or both; as a data type it is an indexed set of vectors. The index can be one-dimensional as in the key-frame number for a video, or it can be multi-dimensional as in the orthonormal bounding box coordinates covering an image segment. An indexed distribution is a local pattern such as the intensity profile of a region of interest, and can be derived from a collection of b-dimensional vectors by introducing an index. A graph represents relational information, such as the relative spatial position of two regions of interest in an image. We do not consider a graph as a primary type of interest, because it can be implemented in terms of the other four data types, with some application-dependent rules of interpretation (*e.g.* transitivity of spatial predicates, such as *left-of*). Therefore, we shall leave it outside the scope of this report.

It follows from the foregoing discussion that vectors form a uniform base type for features representing image content. The primary data type in the Virage Engine is a (indexable) collection of vectors. The primary operators on this abstract data type are:

- **create collection -** This operation creates an empty collection of vectors. The specific form of the collection (e.g., list, set, multiset) is dependent on the implementation.
- **create vector -** This operation takes a collection, an image and a function as arguments and extracts a specific feature vector from the image. The computed vector is placed in the named collection.
- **extract -** This is a generic operation to access an element of the collection. Its arguments are the collection and a user-specified function to specify the element of interest in the collection. This is a non-destructive operation and leaves the initial collection unaltered. This function can utilize any indexing schemes available to the application.
- **distance -**This operation compares two vectors and returns a measure of the distance between them. It is required that each vector of a collection has a corresponding distance function for comparison.
- **combine -** This operation creates a new vector by combining two given vectors with an admissible user-specified function.
- **delete vector -** This operation is required to free the memory associated with a particular vector.
- **delete collection -** This operation deletes the collection from transient and/or persistent memory.

8.2 Primitives

In terms of the Virage Engine, a collection of vectors representing a single category of image information is called a primitive. A primitive is a semantically meaningful feature of an image. Thus color, texture, and shape are all general image primitives. Of course, not all primitives will be applicable across all images. For instance, a color primitive may have no relevance with respect to X-ray imagery. In practice, a primitive is specified as a 6-tuple of the following values:

- **Static information**
  - **primitive_id** - a unique primitive identifier
  - **label** - a category name for the primitive
- **Data retrieval functions**
  - **compute_function** - This function accepts the image data and computes its visual feature data and stores it in a buffer.
  - **distance_function** - This function returns the similarity score for its associated primitive. The query operations of the engine call this function with two data buffers (previously created with *compute_function()*) to be compared. The score which is returned must be in the range from [0.0...100.0]. For maximum discrimination, the spectrum of distances returned for this primitive should be spread over this range evenly or in a reasonably smooth distribution.
- **Data management functions**
  - **swap_function** - The engine takes full responsibility for handling the byte order difference between hardware platforms for easy portability. This allows data that is computed on a certain platform to be easily used on any other platform, regardless of byte-order differences. Each primitive supplies this function which will do the byte-order conversions for its own data. The engine will automatically use this function when necessary, to provide consistent performance across any platform.
  - **print_function** - This function is used to print out the desired information of the associated primitive.

After a primitive is defined, it is registered with the Virage Engine using the *RegisterPrimitive()* function. In addition to the above primitive information, an estimated cost of comparison may also be supplied for the primitive, to aid in query optimization performed by the engine.

8.3 Schema definition

A Virage Engine schema is defined as a 2-tuple: a schema id, and an ordered set of primitives. Similar to primitives, the Virage Engine is notified of a new schema by a *RegisterSchema()* function. The primitive IDs referenced here must have previously been defined using *RegisterPrimitive()*, or must be one of the built-in primitives (see Section 8.4.2). The order in which the primitives are referenced dictates the order in which their functions are called during feature extraction (but not during query processing). This allows primitives to work synergistically and share computational results. A single application is allowed to define and use multiple schemas. The Virage Engine operates as a stateless machine and therefore does not manage the data. Hence the calling application manages the storage and access of the primitive data computed from any schema. The following paragraphs describe the types of operations available to applications utilizing the Virage Engine.

Before an application can determine the similarity between an image description and a set of candidate images, the images must be analyzed by the engine. The resulting feature data is returned to the caller to be used in subsequent operations. Naturally, if an image is to be a candidate image in future operations, the feature vector should be stored in a persistent manner, to avoid re-analyzing the image.

- **analyze_image** - This function accepts a memory buffer containing the original image data. It performs an analysis on the image by invoking the analysis functions of each primitive. The results of this computation are placed in memory and returned to the caller, along with the size of the data. Maintenance and persistent storage of this data is the caller's responsibility. Eventually, these structures are passed into the image comparison entry points.

- **destroy_features** - this function is used to free the memory associated with a visual feature that was previously returned from *analyze_image()*. Typically, this is called after the application has stored the data using the associated persistent storage mechanism.

Any image retrieval application requires the ability to determine the similarity between the query description and any of the candidate images. The application can then display the computed similarity value of all of the candidate images, or convey only the *most similar* images to the user. To do this, *similarity scores* are computed by the engine for the relevant candidate images. An application will call the *comparison* functions provided by the engine. These functions will return a *score structure*, which indicates the similarity between the images being compared. The score structure contains an overall numerical value for the similarity of the two images, as well as a numerical value for each of the primitives in the current schema. This allows applications to use the values of the individual primitive comparisons, if necessary. Following are some of the functions relating to the management of the score structures.

- **create_scores** - This function is used to create a scores structure for later use by the function *compare_into_scores()*. It is a cache of primitive level scoring information that can be efficiently reused to compute a new overall score given a new set of weights. The application is responsible for managing the returned score structure. The function *destroy_scores()* must be called to deallocate the memory for this structure.

- **get_score** - This function is used to access individual score values from the score structure. It provides the mechanism for retrieving the scores for individual primitive values within the structure.

- **refresh_scores** - This function will compute an overall visual similarity distance given a cached scores structure (returned from *compare_into_scores()*) and a new set of weights. This computation is very efficient compared to re-computing all of the score data from the feature vectors.

- **destroy_scores** - This function deallocates a previously allocated score structure created by *create_scores()*.

When two images are compared by the engine, each of the primitives in the current schema are compared to give individual similarity values for that primitive type. Each of these scores must then be used to provide an *overall score* for the comparison. In certain situations, these individual primitive scores may need to be combined differently, depending on the desired results. By altering the ways these individual scores are combined, the application developer has the ability to indicate relative importance between the various primitives. For example, at times the *color distribution* of an image will be much more important than its *texture* characteristics. There may also be cases where only some of the available primitives are required in order to determine which images should be considered the *most similar*. Applications are given flexibility in how the overall score is computed through use of the *weights structure*. The application has control over the weight values for any given comparison through the weights structure, and the following functions:

- **create_weights** - This function is used to allocate a weights structure for use in the compare functions. The associated schema_id will determine the specific format of the structure.

- **destroy_weights** - This function is used to free the memory previously allocated with *create_weights()*.

- **set_weight** - This function sets the weight in the weights structure identified by the given primitive_id, which identifies the primitive whose weight is to be set. The value should be a positive floating point number. In general, weights are normalized before use by calling *normalize_weights()* .

- **get_weights** - This function is used to extract an individual weight value from a weights structure.

Every application will have unique requirements in the way they determine which images are to be considered most similar, and how to efficiently manage a changing set of results. Certain applications may need to do an exhaustive comparison of all images in the candidate set, while others are only "interested" in a certain set which are most similar to the query description. Certain applications (or situations) may also require the ability to quickly manipulate the relative importance of the primitives, using the individual primitive scores and weights, as discussed above. Some of the more common comparison functions which address these requirements are described below:

- **compare** - This is the simplest entry point for computing the overall visual similarity for two given images, represented by their respective visual features. The caller passes in a weights structure and *compare()* computes and returns the weighted overall score, which is a numerical value in the range [0.0...100.0]. This function can be used when a score is required for every candidate image. If only the top *N* are required, the function *compare_less_than()* may be more appropriate. Subsequently, if scores are desired for additional sets of weights, then use *compare_into_scores()*.

- **compare_less_than** - This function can be used for optimized searches in which the scores of every single candidate image are not required. A threshold similarity distance is passed in to indicate that any image whose score is above the threshold is not of interest for this search. As soon as the engine determines that the image is outside this range, it terminates the similarity computation and returns a flag to indicate that the threshold has been exceeded. This provides a significant performance boost when *top N* style searches are sufficient. Again, it is the application's responsibility to determine the appropriate threshold value for each comparison.

- **compare_into_scores -** This function also performs comparisons, but it returns a scores data structure that caches scoring information from each primitive. The score data, like the visual feature data, must be managed by the application. It can be used effectively in situations where a ranking of candidate images is desired for several sets of weights. One companion function to this is the *refresh_scores()* function which can quickly compute a new overall score given one of these score structures and a desired set of weights. It is also possible for the developer to use *get_score()* to extract component scores from the structure and perform a custom combination of these to achieve a final score.

8.4 The Search Engine

8.4.1 Primitive design

The "pistons" of the Virage Engine are the primitives. A primitive encompasses a given feature's representation, extraction, and comparison function. In addition to the formal engineering requirements discussed in Section 8.2, there are a number of heuristics which lead to effective primitive design. These design constraints are not hard rules imposed by the engine architecture, but rather goals that lead to primitives which are "well-behaved". For a given application, an engineer may choose to intentionally relax certain constraints in order to best accommodate the tradeoffs associated with that domain.

- **meaningful -** Primitives should encode information which will be meaningful to the end-users of the system. Primitives, in general, map to cognitively relevant image properties of the given domain..
- **compact -** A primitive should be represented with the minimal amount of storage.
- **efficient in computation -** Feature extraction should not require an unreasonable amount of time or resources.
- **efficient in comparison -** Comparison of features should be extremely efficient. The formulation should take advantage of a threshold parameter (when available), and avoid extraneous processing once this threshold has been exceeded. The distance function should return results with a meaningful dynamic range.
- **accurate -** The computed data and the associated similarity metric must give reasonable and expected results for comparisons.
- **indexable -** The primitive should be indexable. A secondary data structure should be able to use some associated value(s) for efficient access to the desired data.

8.4.2 Universal primitives

Several “universal primitives” are included with the Virage Engine. These primitives are universal in the sense that they encode features which are present in most images, and useful in a wide class of domain-independent applications. Each of these primitives are computed using only the original data of the image. There is no manual intervention required to compute any of these primitives. A developer can choose to mix-and-match these primitives in conjunction with domain specific primitives (see Section 8.4.3) in order to build an application. These primitives have been designed based on the above heuristics.

- **Global color** - This primitive represents the distribution of colors within the entire image. This distribution also includes the *amounts* of each color in the image. However, there is no information representing the locations of the colors within the image.
- **Local color** - This primitive also represents the colors which are present in the image, but unlike *Global color*, it emphasizes *where* in the image the colors exist.
- **Structure** - This primitive is used to capture the *shapes* which appear in the image. Because of problems such as lighting effects and occlusion, it relies heavily on shape characterization techniques, rather than local shape segmentation methods.
- **Texture** - This primitive represents the low level textures and patterns within the image. Unlike the *Structure* primitive, it is very sensitive to high-frequency features within the image.

8.4.3 Domain specific primitives

Applications with relatively narrow image domains can register domain specific primitives to improve the retrieval capability of the system. For applications such as retinal imaging, satellite imaging, wafer inspection, etc., the development of primitives that encode significant domain knowledge can result in powerful systems. Primitives should obey the design constraints of Section 3.1, but there is considerable flexibility in this. For example, a wafer inspection primitive may be designed to look for a specific type of defect. Instead of an actual distance being returned from the distance function, it can return 0.0 if it detects the defect, and 100.0 if not. In addition, primitives can provide their own "back door" API's to the application developer, and expose parameters that are controlled independently from the weights interface of the Virage Engine. There is also ample opportunity for a set of domain primitives to cooperate through shared data structures and procedures (or objects) in such a way that they can economize certain computations and information. Figure 8 shows some of the key components of the Virage Image Search Engine.

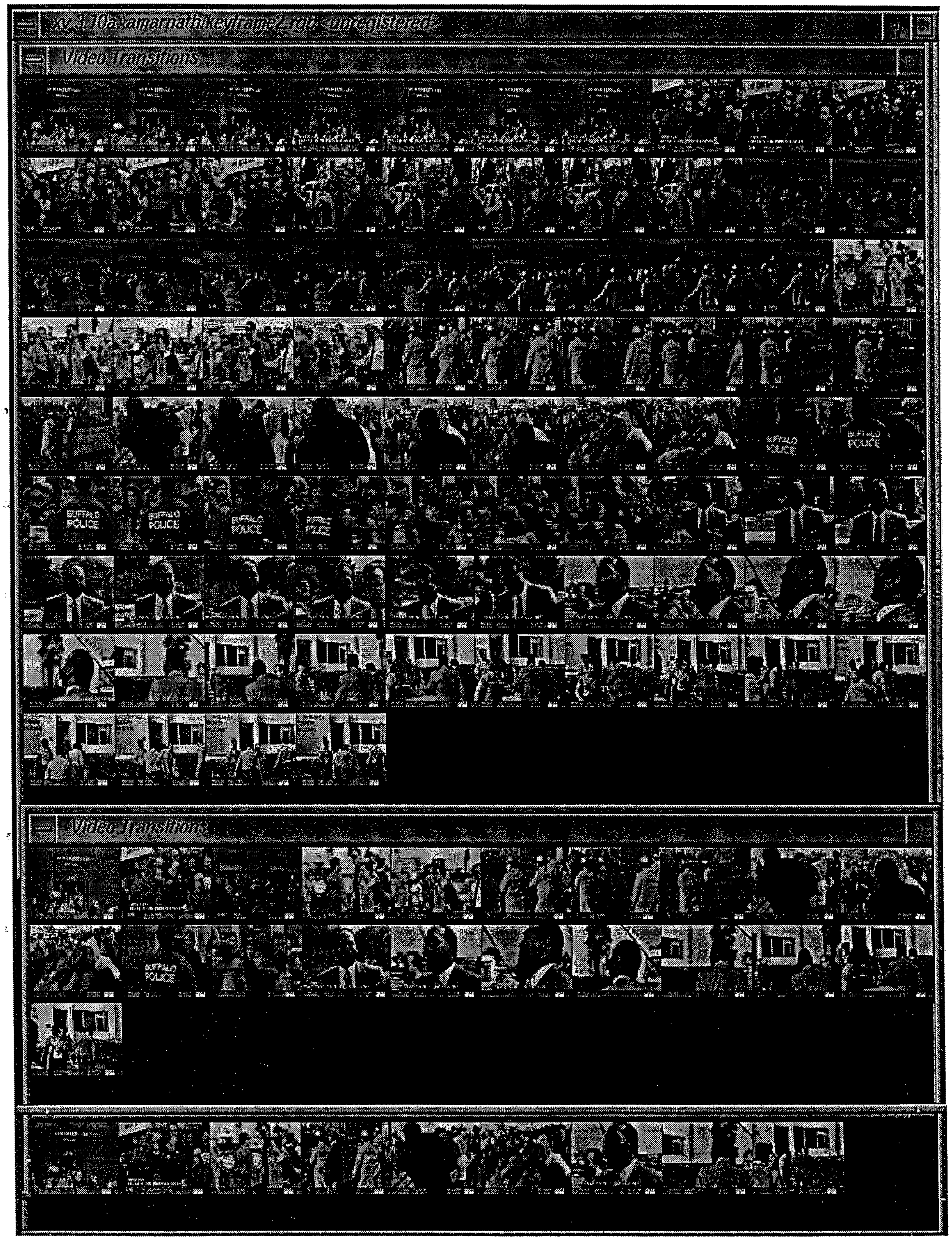


Figure 4: Two different sets of customized keyframe sets have been created from the same base keyframe set. By the current design the user application must experimentally determine what is a proper summary.

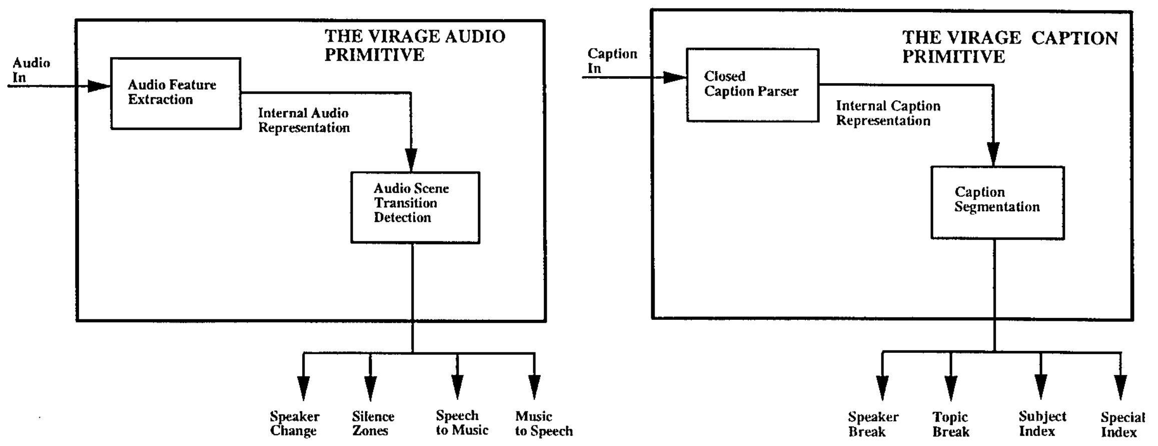


Figure 5: **Left:** Audio Primitive **Right:** Caption Primitive

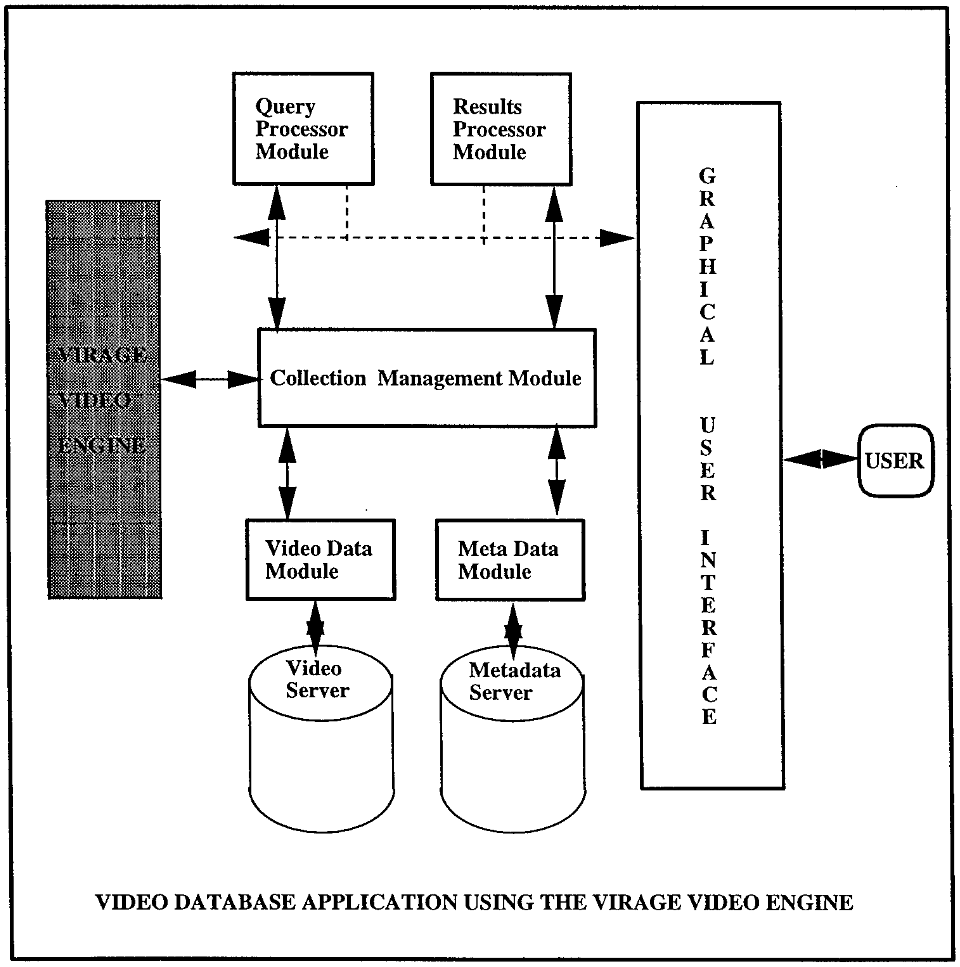
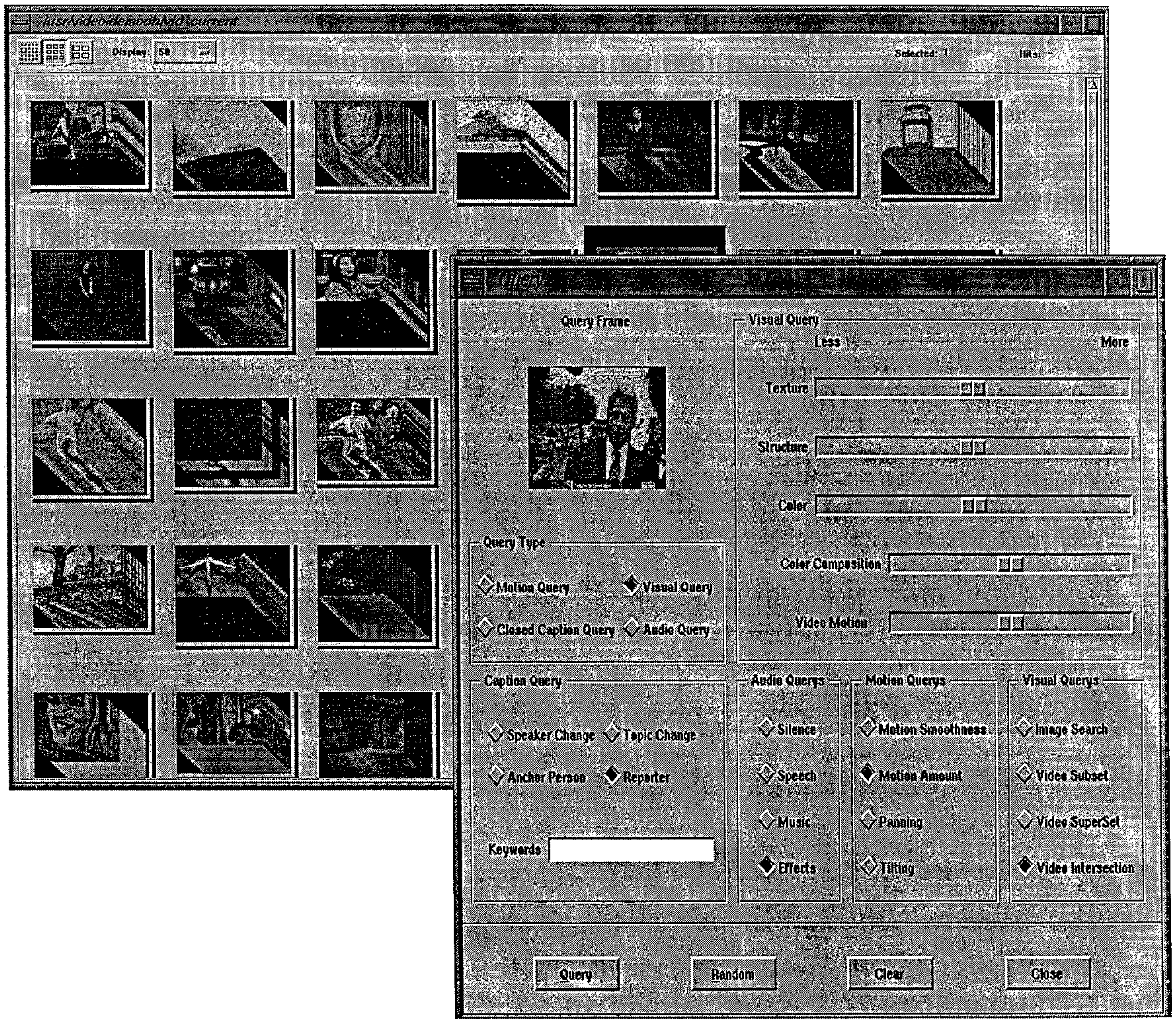


Figure 6: Video Database using the Virage video engine



Query Type: ◇ Motion Query [✓ Visual Query] ◇ Closed Caption Query ◇ Audio Query

Caption Query: ◇ Speaker Change ◇ Topic Change ◇ Anchor Person [✓ Reporter]

Keywords

Audio Querys: ◇ Silence ◇ Speech ◇ Music [✓ Effects]

Motion Querys: ◇ Motion Smoothness [✓ Motion Amount] ◇ Panning ◇ Tilting

Visual Querys: ◇ Image Search ◇ Video Subset ◇ Video SuperSet [✓ Video Intersection]

Figure 7: Graphical User Interface for a prototype video database

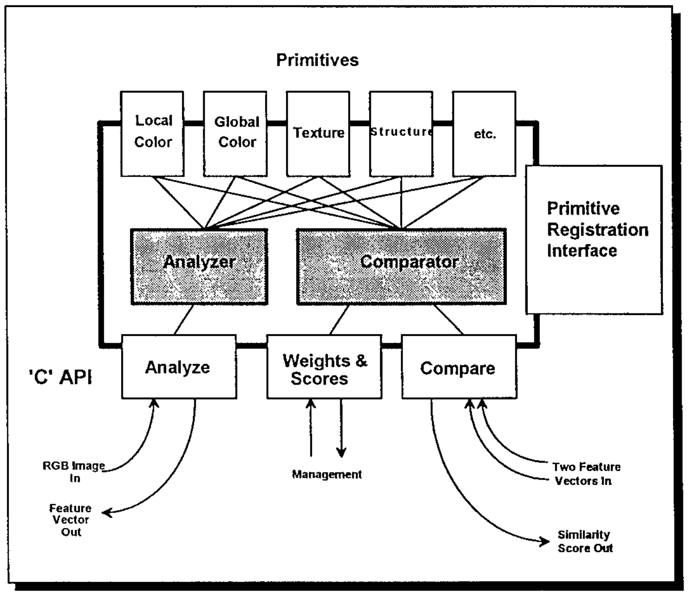


Figure 8: Components of the Virage Image Search Engine.